2002-01-2955

Formulation of an Integrating Framework for Conceptual Object-Oriented Systems Design

Jack Zentner [1], **Peter W. G. De Baets** [2], **and Dimitri N. Mavris** [3]
Aerospace System Design Laboratory
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta GA 30332-0150

ABSTRACT

In this paper, an overview is given of the different alternatives to an integrating computational framework. A new framework will be introduced, which incorporates the latest computational techniques and more importantly a mind-set emphasizing flexibility, modularity, portability and re-usability.

Distributed object computing extends an object-oriented system which allows objects to interact across heterogenous networks and interoperate as a unified whole. Integrated computing frameworks are discussed, together with data transport techniques such as Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP) to achieve platform, code and meta-model independent integration.

In addition, the paper will illustrate through an air vehicle example that using open-source tools are a valid alternative to commercial packages. Added advantages are the access to source code which is extremely useful in a conceptual systems-of-systems research environment.

WHY INTEGRATING FRAMEWORKS?

Today's engineering designers have come to the realization that no longer can successful designs revolve around the analysis and optimization of a single discipline. But rather, successful designs are now viewed as a balance between competing disciplines. Given the above, accounting for and balancing disciplines through the sharing of data between disciplines becomes a monumental task.

To date several commercial applications and research programs [1] [2] [3] have been developed to aid in the dissemination of information between discipline analyses. Nonetheless, these tools do not always afford the designer the flexibility necessary to implement novel information distribution and manipulation techniques.

The requirements for this research stem from the needs of MDO (Multidisciplinary Design Optimization). The term MDO was coined almost a decade ago. This relatively new field consists of the following principal conceptual components [4]:

1. Design Oriented Analysis: System level designing allows the designer to answer the "what-if" questions. Designers want to know the sensitivity of the design with respect to the design variables.
2. Approximation Concepts: Meta-models allow the designer the ability to bypass the expensive

[*] Graduate Research Assistant
[†] Graduate Research Assistant
[‡] Professor, Director ASDL, Boeing Chair in Advanced Aerospace Systems Analysis

direct coupling of analysis codes to the design space explorer tool. Common meta-modeling techniques such as Response Surface Methodology (RSM) and Neural Networks (NN) can be used instead of these complex disciplinary analysis codes.

3. Mathematical Modeling of System: It is axiomatic that an engineering system is usually modeled by multiple disjoint analysis codes and not one monolithic code. Data reduction techniques may need to be applied if large amounts of data are exchanged between codes.

4. Decomposition: Given that codes analyzed on the same level are often tightly coupled, it is usually preferable (if possible) to decouple the individual codes and let the system-level take care of the coupling. Here system decomposition techniques and tools such as the Bi-level Integrated System Synthesis (BLISS [5]) are used.

5. Design Space Search: Exploration of the design space is the search to find the constrained minimum. Various algorithms such as Sequential Linear Programming (SLP) and Sequential Quadratic Programming (SQP) can be applied. Alternatively, where applicable, algorithms employing stochastic processes (genetic algorithms (GA) and simulated annealing (SA)) are also an option.

6. Optimization Procedures: System optimization is conducted at the system-level. The system-level optimizer knows which codes to execute and in what fashion. This element effectively ties together the different codes in an execution sequence.

7. Human Interface: Manual intervention in the system design process is important and is not an after-thought. In a well designed environment, the implementation should allow for straightforward, designer intervention. This intervention is needed since MDO often relies on human interaction to guide the process.

Typically each analysis code handles one disciplinary component of the overall systems engineering problem. This implies that at the top-level all these disparate codes, each with unique data formats and running on different platforms, need to communicate with each other through some system-level executer/controller. Furthermore, there is often a coupling of inputs and outputs between analyses resulting in an iterative analysis loop.

The problem is formulated by Sobieszczanski-Sobieski [4] as follows: engineering system analysis is expensive, time-consuming and a non-trivial managerial task. Hence the clear need for a framework to handle distributed MDO problems. Such an integrating framework needs to capture all the enumerated MDO components if the implementation is to be successful.

HISTORY OF INTEGRATING FRAMEWORKS

Frameworks try to aid engineers in formulating, solving and evaluating complex design problems. Automation of the design process occurs through the framework.

HARD-CODED ALGORITHMS

In the earliest frameworks, all disciplinary executables were brought together and execution control was given to a fixed algorithm. There are three variations on this in chronological order: monolithic codes, direct integration, and meta-modeling techniques. Most of the monolithic programs were written in FORTRAN and some examples of these efforts are still around, for example: FLOPS (Flight Optimization System) and ACSYNT (Aircraft Synthesis). A general disadvantage of these systems is the relative difficulty to include higher-fidelity tools as they become available. Since these approaches require a total reconfiguration of the script that controls the execution.

Up to recently, there was no real valid alternative to this *hard-coding* of programs. In the last decade, some commercial alternatives arose on the horizon.

COMMERCIAL APPLICATIONS

In the design and analysis of systems, there are currently several design tools available that allow a designer to efficiently explore and design with the overall system in mind. Commercial efforts have produced AML, iSIGHT, and ModelCenter. These allow for a textual or graphical representation of the data flows between analysis tools, automatic output parsing, input file generation, optimization tools, statistical tools, and result visualization.

However, these environments have certain, inher-

ent drawbacks. Most notable, they are not open-source thus not allowing the designer to tailor the tool to exactly meet the needs that the analysis may require. These tools allow for the designer to link codes through the GUI (Graphical User Interface), but do not always give direct access to the underlying core of the tool. Consequently, they are not truly conceptual design tools since they do not allow the investigation of concepts. They only allow for perturbations around a user-provided baseline input file.

In that sense, these commercial packages are preliminary tools. To use these packages at the conceptual level is possible, however requires extensive changing and use of the sometimes provided API (Applications Programmer's Interface). As a result, they become not much different from an actual programming language.

An interesting comparison can be made by looking at Microsoft Excel and The MathWorks' Matlab. Excel inherently uses a GUI, the spreadsheet, to enter equations and visualize its output. Matlab on the other hand opens up a library of functions, which can be used from a command input window. Over time, Excel added the capability of VBA (Visual Basic for Applications), which allowed for more powerful operations comparable to Matlab. Nonetheless, Matlab is still a more powerful and flexible tool since it was conceived as an API. The same comparison is true for integrating frameworks: most commercial applications use a GUI to interact with the user. It is this GUI which makes these tools very user-friendly and easy to use, however, a general API built on solid O-O (Object-Oriented) programming is potentially much more powerful.

OPEN-SOURCE APPLICATIONS

An alternative solution is to develop a general, systems analysis API. Such a general library of methods would allow the designer to programmatically link and execute any number of analyses and manipulate the resulting data in any conceivable manner [6] [7].

Recent advances in business-to-business data transfer as well as server-client application interfacing have made the task for the engineer to develop such an environment significantly easier. More specifically, it is straightforward to create an API that incorporates O-O code wrappers written in Java with uniform, standard data transport tools (XML (Extensible Markup Language) and SOAP (Simple Object Access Protocol)) to create an infrastructure which is both platform independent and flexible to the needs of the designer.

Starting such a task from scratch would entail a significant programming feat for any designer/programmer. Fortunately, readily available tool boxes for optimization [8], statistics [9] [10], simulation, visualization [11], and web-server applications [12] are pre-written, plug-and-playable, and more importantly, open-source freeware.

These framework tools combined with the use of the agents give total flexibility and modularity. This allows the designer to concentrate on the actual design task. More details on the open-source building blocks will follow in later sections.

COMMERCIAL APPLICATIONS

Scott [13] gave a good overview of the commercial endeavors. The following high-level overview of three commercial packages is given here as an introduction.

ADAPTIVE MODELING LANGUAGE

AML (Adaptive Modeling Language) is developed by Technosoft [14]. AML is built on the philosophy of O-O software design and uses LISP as its programming language, which is a fairly uncommon language. Variables are created by instances of some previously defined primitives. When defining formulas, AML automatically keeps track of which variables depend on others. AML has easy and polyvalent graphical visualization capabilities (especially for aerodynamic design).

Some disadvantages from a user friendliness perspective are that a fairly good working knowledge is required of O-O programming. The use of O-O programming is not necessarily detrimental as will be shown when discussing the open-source requirements. Unfortunately, integration with certain tools (Excel spreadsheets etc.) is not functional yet.

ISIGHT

iSIGHT is produced by Engineous Software [15]. iSIGHT is based on MDOL (Multidisciplinary Optimization Language), its own language. Pre-made building blocks are accessible through GUIs so

to avoid direct interaction with the underlying language. Logic-based control and optimization boxes are readily available from the GUI environment. Options for parsing input and output files are very extensive. The linking between codes occurs implicitly by using the same variable names. Unfortunately, cross-platform integration of different codes and front-end is not straightforward.

MODELCENTER

ModelCenter is made by Phoenix Integration [16]. The front-end interface is called ModelCenter, while in the background the Analysis Server services the request coming from the ModelCenter GUI [17]. Using the ModelCenter "*web-browser*", it is very easy to use a resource/code once it is wrapped and placed on the Analysis Server from any location.

Response Surface generators, Monte Carlo simulation and stochastic optimization toolboxes were recently added to the basic package. One of the remaining drawbacks of ModelCenter is that multiple instances of a code, also known as parallelization, is currently not supported.

Given the easy of use and polished execution that these commercial packages exhibit, there are drawbacks as well. Most notability, the proprietary nature of the source code makes customization difficult. In-house developed methodologies and strategies can be integrated, but that requires clever interaction through a GUI or the user has to use the provided API in a programmatic environment. Access to source code for extreme flexibility is a very important prerequisite in a conceptual design research environment where these new methodologies are developed and investigated.

OPEN-SOURCE APPLICATIONS

The previous section highlighted some of the advantages and disadvantages of commercial packages. An open-source tool should can draw on the strengths and weaknesses listed above. Below are a list of items that are worthy for incorporation and investigating in this to-be created tool.

- Use the sound basis of O-O programming (from AML).
- Extensive tools for parsing input and output files (from iSIGHT).
- Logic-based control boxes are pre-written and available as functions (methods in Java) and in the API (from iSIGHT).
- Wrapped codes (also called *agents*) are immediately available form a distributed servers (from ModelCenter).
- Methods for multiple instances of agents for parallelization need to be provided (from a shortcoming in ModelCenter).
- Allow for growth potential when incorporating statistical (from ModelCenter), optimization (from iSIGHT), and visualization toolboxes (from AML).

OBJECT-ORIENTED BUILDING BLOCKS

Distributed object computing extends an O-O system which allows objects to interact across heterogenous networks and interoperate as a unified whole. Integrated computing frameworks are discussed, together with data transport techniques such as XML and SOAP to achieve platform, code and meta-model independent integration.

Extensible Markup Language

XML is a meta-markup language for text documents. The data is included as strings of text marked-up by tags describing the data. There are two important features to XML which make it very useful in data transfer [18] [19].

Firstly, portability. Just like Java, XML is portable since it is merely a text file and can be directly transferred between platforms. Java and XML produce "portable code, portable data."

Secondly, interoperability. The XML standard [20] specifies the format and structure of an XML file but not the content of the tags, the strings, the attributes, etc. An XML file can define airplane data as easily as it can contain a conference paper, as long as it conforms to the formatting standards.

Simple Object Access Protocol

Like XML, SOAP is a standard [21]. SOAP allows for straightforward data transfer using HTTP (Hyper-Text Transfer Protocol) as the transport layer. The use of HTTP helps to resolve complicated issues as firewalls, ports, sockets, etc.

There are two implementations of the SOAP standard: Apache [22] and Microsoft. The Apache implementation specifies two methods to invoke SOAP services: RPC (Remote Procedure Call) and the message-based model. The former is used in this research, and the model can be described with the following steps [23]:

1. A client builds an XML document specifying the server which will service the request, the name of the method and associated parameters used by the method.
2. The server decodes the received XML document and executes the method.
3. After execution the results are packed in a XML document and returned.
4. The client decodes the XML response.

Agents and Models

Earlier work by Hale and Mavris [24] documented the history of frameworks and the evolution to the new *collaborative* environment. All these frameworks try to aid the engineer in complex design problems with solutions that require analysis from several domains. The automation of the solution process requires control and communication of domain analysis be provided by the framework [25] [26].

Within this framework, a key technology is the implementation of the *agent*. Hale and Craig [3] added a new component to the agent: the *model*, and proposed an updated definition:

> An agent is a resource, which has been modeled and wrapped for inclusion in a distributed design environment. The agent design requires a designer-centered, bi-directional wrap, independent of proprietary boundaries and capable of supporting increasing fidelity models.

These agents can generate accountable design information. This includes the "what, why, when, and where" information needed as decision-support for the designer. The result is intelligent agents, which effectively conceal the proprietary codes and data formats from the end user. The schematic in Figure 1 depicts the breakdown of these agents.

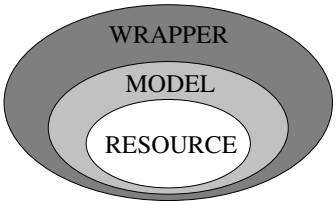

Figure 1: Agent Components

The wrapper is a bi-directional information exchange layer which shields the resource and model from the computing backplane. The main role of the wrapper is data exchange and conditioning.

The second element, the model, adds context to the information provided by the agent. Models include behavior and implementation information. The former is a mathematical formulation, engineering principle, or geometrical construction describing explicitly what the resource does. The latter captures execution characteristics: variable definition, file descriptions, units, executions characteristics, and platform dependencies.

Using Java, this model is easily generated by the Javadoc utility. Javadoc is the tool for generating API documentation in HTML format from "doc" comments in source code. These "doc" comments include, but are not necessarily limited to, identification of parameters and methods provided by the wrapper and resource.

Lastly, the resource is the computer program. Typically these are off-the-shelf analysis codes. Examples include ASTROS, ANSYS etc.

APPLICATION

FRAMEWORK COMPONENTS

The distributed framework can be easily divided into two separate collections of classes, the client API and server API. The client API is the library of classes which the user directly programs with when performing a systems analysis. This API is primarily made up of two types of classes, the Design Space Explorer classes and the Code classes.

DesignSpaceExplorer objects implement fundamental systems analysis functionality. For instance, a DoeDSE object is able to conduct a design of experiments analysis around any Code object. The Op-

timizerDSE, is a design space explorer object that has the ability to run various optimization algorithms (SQP, SLP, GA, etc.) on any Code object. Obviously, a new design space explorer class could be created to perform any conceivable Code execution algorithm.

The Code objects are the front end of the bridge between the client computer and the server running the Agents objects. To the user, a Code object is merely a local implementation of their favorite code. They need not have any idea as to how or where the actual code is implemented.

On the server side, there is a one-to-one mapping between the Code objects available on the client side and the corresponding Agent objects. Agents are wrappers for a legacy command line executable codes. Each Agent is comprised of an InputFileUpdater object, InputFileBuilder object, an Executor Object and an OutputFileParser object. In general, the input file builder, executor and output file parsers classes will be unique for each wrapped legacy code. Since XML is the native file format and all the client Code objects send a similar type of XML file, the underlying InputFileUpdater implementation is the same for all agents.

Figure 2 is a simplified UML (Unified Modeling Language) class diagram. An important observation of this figure shows that there is minimal concrete class to concrete class interaction. Meaning that the framework was built with extensibilty in mind by requiring that the developer couple classes abstractly (through the use of inheritance from interface classes) rather than allowing direct concrete to concrete class interactions.

PARAMETRIC AIRCRAFT

As an example implementation, the choice was made to use FLOPS. A baseline aircraft file was used as input. The airplane represented in this file was a generic 150 passenger short-range airplane. Some variables used are wing aspect ratio (AR), maximum cruise altitude (CH), thrust-to-weight ratio (TWR), taper ratio of the wing (TR), quarter-chord sweep angle of the wing (SWEEP), wing thickness-chord ratio (TCA), and factors to in- or decrease fuel flows (FACT), lift-independent drag (FCDO), and lift-dependent drag (FCDI).

The authors acknowledge that the above implementation using open-source components is also easily achieved with the commercial packages. The current problem was used to illustrate the relative ease of achieving similar functionality and the same time showing that the limits to the flexibility of the framework are unlimited.

PROCESS FLOW

A top-level overview of the process is given in Figure 3.

The entire process consists of eleven steps which will be discussed in more detail now.

1. The variable ranges are entered in Excel spreadsheet. To allow parsing and manipulation of this information it needs to be converted into an XML format. Similarly, a design array was created in external programs and is saved in a tagged (XML) format to allow easy use of this information in further steps. This is where the designer comes up with his own tagging scheme which defines the data and allows other users to understand what is contained in the files. This step is illustrated in Figures 4 and 5.
2. The two above files are then merged into multiple different XML files which each specify a certain run in the Design of Experiments. Each file has a certain actual (non-normalized) setting of the variables (high, nominal or low). This is what is depicted in Figure 6.
3. The XML-formatted design run files are sent to the server where FLOPS is residing through a SOAP protocol. The internal workings of SOAP were explained in the previous section.
4. Similar to step 1, the FLOPS baseline text file needs to be converted into a XML format. The conversion to XML format makes a distinction between changing (variables) and fixed information as shown in Figure 7.
5. This step combines the output from the two previous steps 2 and 4 (DOE runs in XML format and macro in XML format) to create updated input files in XML format for each specific DOE run. This is graphically illustrated in Figure 8. This happens on the agent side, i.e. on the server where FLOPS resides.
6. Since FLOPS has no XML input format but a straight flat file (no markup), the updated input

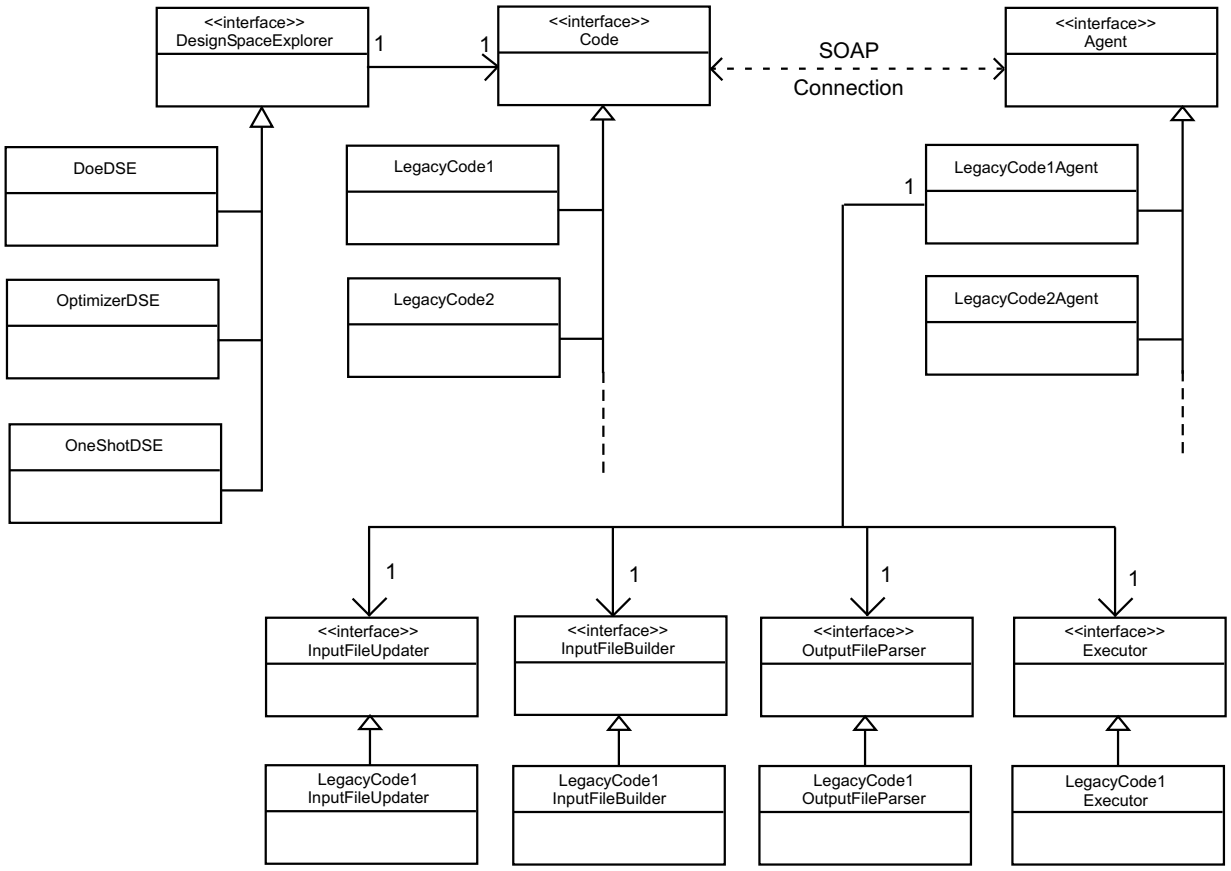

Figure 2: Simplified UML Class Diagram

file in XML format needs to be translated back to a text file.

7. This step just executes FLOPS. The execution is launched remotely by the server with no interaction from the user.

8. Following the execution, the output file is translated to XML format where tags identify the information pertinent to the designer (see also Figure 9).

9. Steps 6,7 and 8 are executed iteratively, i.e. each one of the updated input files goes through this process.

10. The SOAP call is terminated by the sending back of the XML output file.

11. Lastly, after all updated macros are executed and respective responses are collected on the user's computer, all the data is compiled in Excel as shown in Figure 10.

FUTURE WORK

As summed up when describing the needed capabilities of this framework, a couple of areas of future work were identified. From the outset, growth possibilities for incorporating statistical (Response Surfaces, Monte Carlo, stochastic methods etc.), optimization, and visualization capabilities were envisioned. Two major areas of focus identified at this time are optimization and statistical analysis toolboxes.

1. The former will most likely rely on an existing design optimization tool libraries such as Vanderplaats' DOT (Design Optimization Tools). DOT was written in C language and the most efficient way of accessing these functions is through the JNI (Java Native Interface). JNI allows Java code to operate with applications and libraries written in other languages, such as C and C++.

2. The latter is undergoing careful investigation and no choice of tool has been made yet. One tool, JMP [27] is a Windows application developed by the SAS Institute. JMP interacts with the user through a GUI, whom can perform different kinds of statistical analyses with varying levels of output. For framework purposes, this analysis would have to be accessible from a batch mode which is currently the biggest bar-

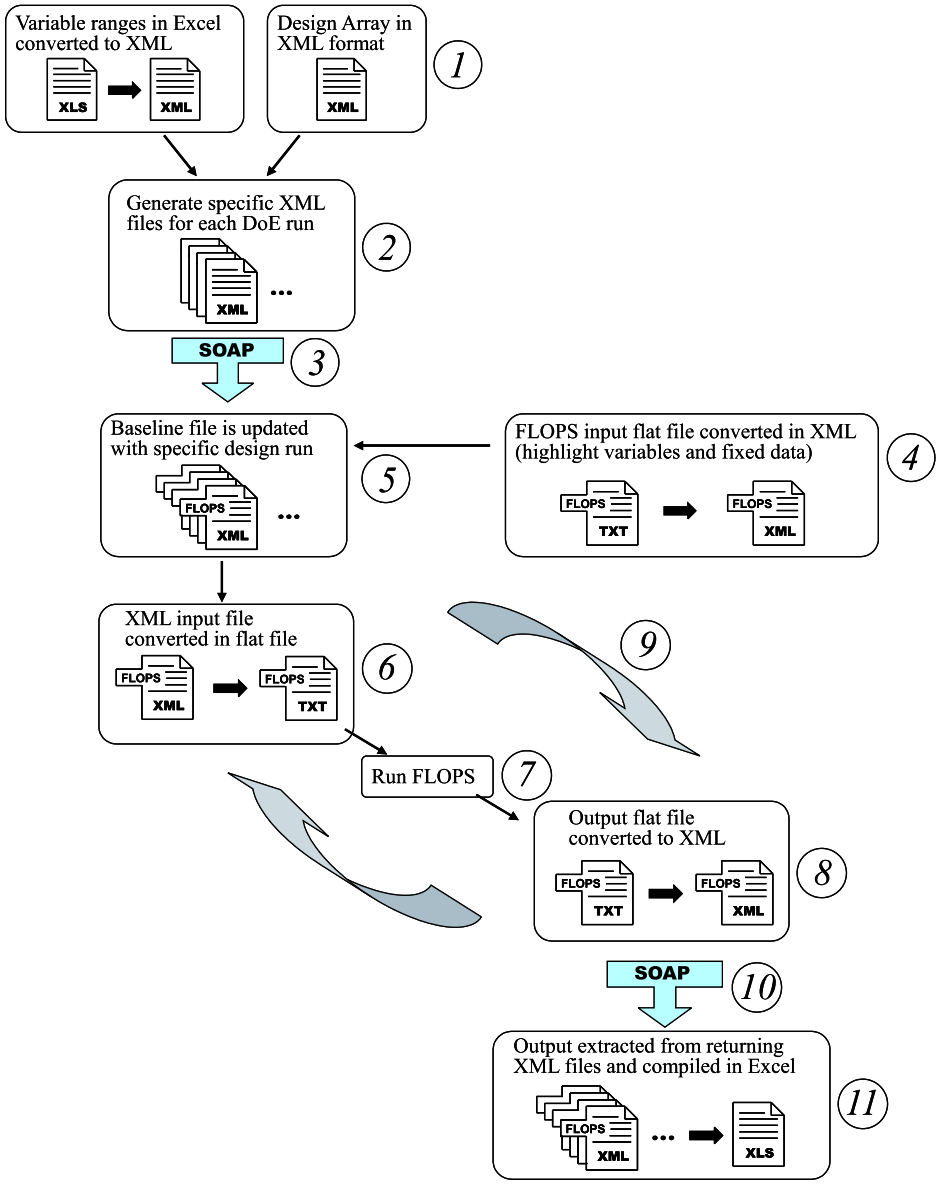

Figure 3: Framework Process Flow

rier to be conquered. Other potential toolbox from the open-source community are also under investigation.

CONCLUSIONS

An overview was given of the different alternatives to an integrating computational framework. The seven conceptual elements of a good MDO environment were identified and it was illustrated that these elements can be captured with a design tool made of open-source elements.

In this paper a backbone computing framework was introduced which incorporates the latest computational techniques and more importantly, a mindset emphasizing flexibility, modularity, portability, and re-usability. With the described object-oriented tools, such a framework can now truly be built for the first time by non-computer scientists.

The paper illustrated that usage of open-source tools were a valid alternative to commercial packages. Added advantages are the access to source code which is extremely useful in a conceptual systems-of-systems research environment in which the authors work. It is important to stress however that for out-of-the-box performance the commercial applications can not be beat. They perform very well and for most problems are the optimal solution.

Growth potential of the open-source framework was allowed for from the outset. Future work was identified focusing on the optimization and statistical capabilities. These improvements will add significant

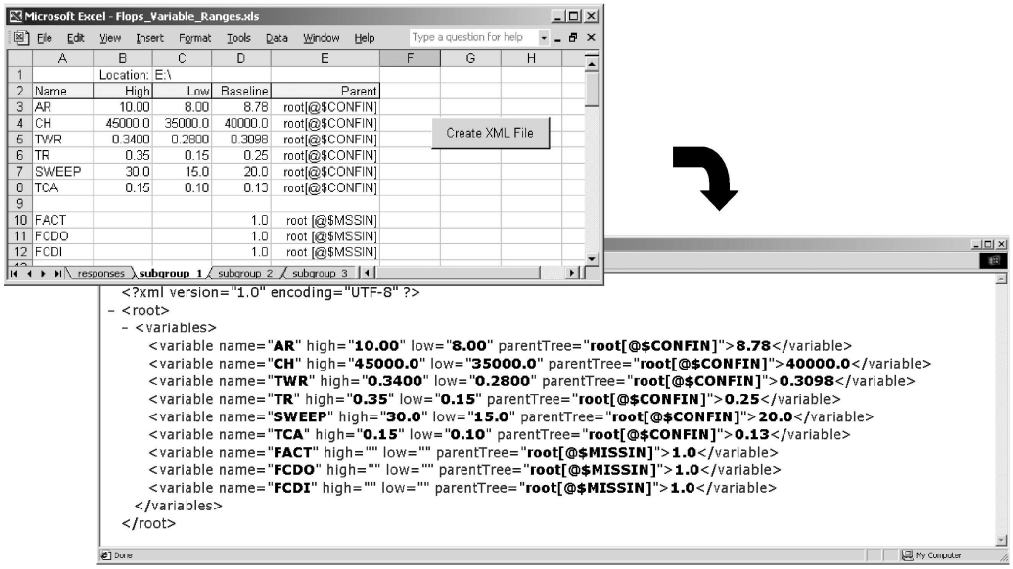

Figure 4: Variable Ranges Converted from Excel To XML Format

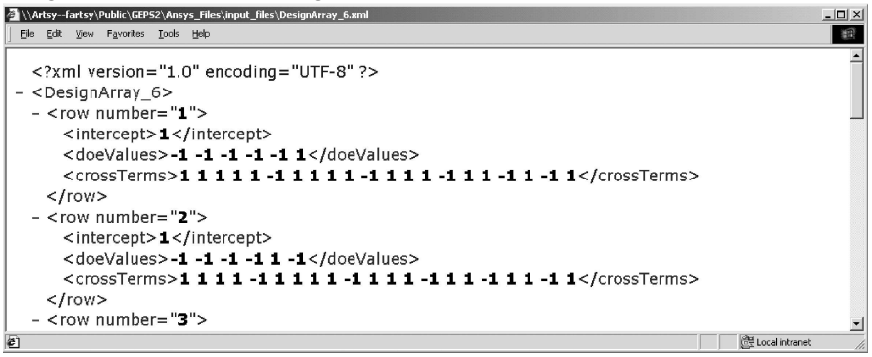

Figure 5: Design Array in XML Format

computing and analytical force and make the comparison with the commercial alternatives much more competitive.

ACKNOWLEDGEMENTS

This research was sponsored by a NASA Langley Research Center Contract under supervision of Mr. Bob McKinley. Dr. Vitali Volovoi for providing insight and generous support. iSIGHT and ModelCenter licenses provided to the ASDL (Aerospace Systems Design Laboratory) courtesy of respectively Engineous Software and Phoenix Integration.

REFERENCES

[1] A. J. Morris, H. Syamsudin, J. P. Fielding, M. Guenov, K. H. Payne, P. J. Deasley, S. Evans, and J. Thorne. MACRO - A tool to support distributed MDO. In *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, number 2000-4702, September 2000.

[2] A. J. Morris. Supporting Distributed Multi-disciplinary Design and Optimization Teams. *Cranfield Univeristy Aerogram*, 10:21–26, March 2002.

[3] M. A. Hale and J. I. Craig. IMAGE: A Design Integration Framework Applied to the High Speed Civil Transport. In *1st Industry/Acedemy Symposium on Research for Future Supersonic and Hypersonic Vehicles*, number HM301, December 1994.

[4] J. Sobieszczanski-Sobieski. Multidisciplinary Design Optimization: An Emerging New Engieering Discipline. In *World Congress on Optimal Design of Structural Systems*. Kluwer, August 1993.

[5] J. Sobieszczanski-Sobieski, J. S. Agte, and R. R. Sandusky Jr. Bilevel Integrated System Synthesis. *AIAA Journal*, 38(1):164–172, January 2000.

[6] K. Knoernschild. *Java Design: Objects, UML, and Process*. Addison-Wesley, 2001.

[7] A. J. Riel. *Object-Oriented Design Heuristics*. Addison Wesley, 1999.

[8] DOT (Design Optimization Tools) Homepage. `http://www.vrand.com/dot.htm`, August 2002.

[9] The R Project for Statistical Computing. `http://www.r-project.org/`, May 2002.

[10] The Omega Project for Statistical Computing. `http://www.omegahat.org/`, May 2002.

[11] VisAD - Homepage. `http://www.ssec.wisc.edu/`, May 2002.

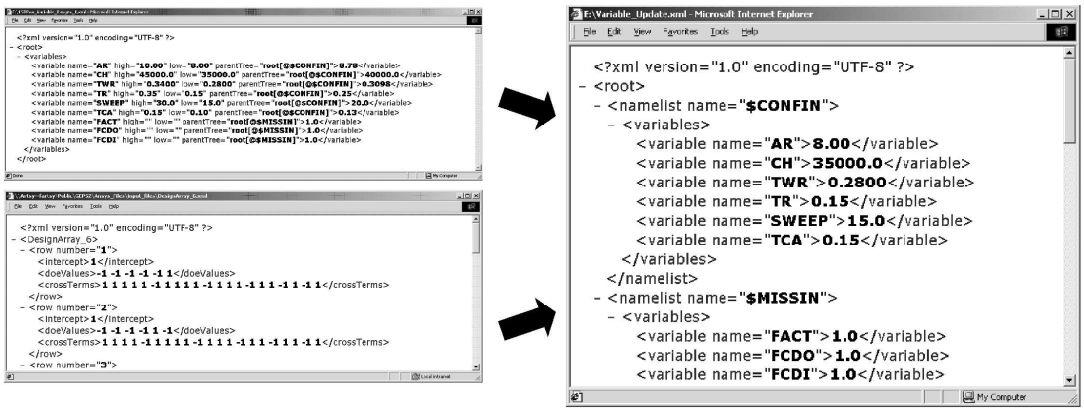

Figure 6: Design Array and Variable Ranges Give Specific Runs of the Design of Experiments

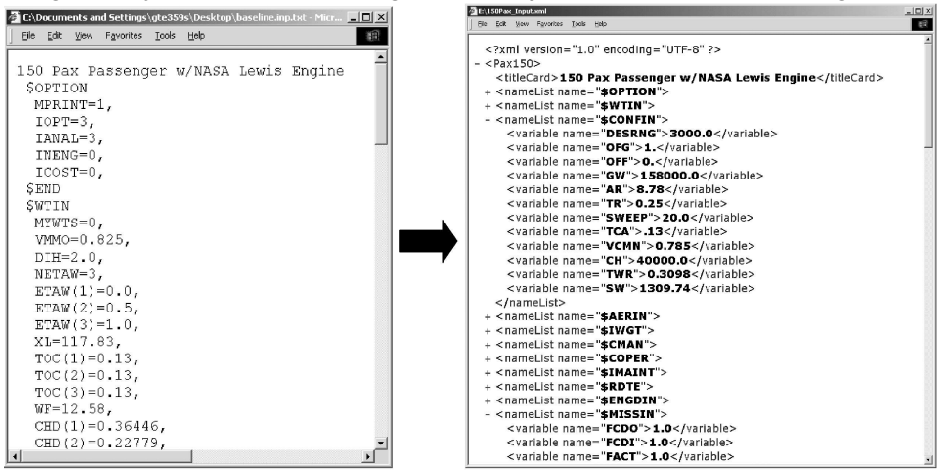

Figure 7: FLOPS Input File Converted from text to XML Format

[12] J. Goodwill. *Apache Jakarta-Tomcat*. Apress, 2001.

[13] A. T. Scott and J. Olds. An Evaluation of Three Commercially Available Integrated Design Framework Packages for Use in the Space Systems Design Lab. Technical report, Georgia Institute of Technology, April 2001.

[14] TechnoSoft Inc. - Homepage. `http://www.technosoft.com/`, August 2002.

[15] Engineous Software - Homepage. `http://www.engineous.com/`, August 2002.

[16] Phoenix Integration: Software for Engineers: Using ModelCenter and Analysis Server Worldwide - Homepage. `http://www.phoenix-int.com/`, August 2002.

[17] Improving The Engineering Process with Software Integration - Integrating Engineering Applications for Design. `www.phoenix-int.com/publications/` , July 2002.

[18] B. McLaughlin. *Java & XML*. O'Reilly, 2001.

[19] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly, first edition, January 2001.

[20] World Wibe Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). `http://www.w3c.org/TR/REC-xml/`, May 2002.

[21] World Wibe Web Consortium. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3c.org/TR/SOAP/`, May 2002.

[22] The Apache Software Foundation. `http://www.apache.org/`, May 2002.

[23] Apache SOAP. `http://xml.apache.org/soap/`, May 2002.

[24] M. A. Hale and D. N. Mavris. Enabling Advanced Design Methods in an Internet-Capable Framework. In *4th World Aviation Congress and Exposition*, number SAE/AIAA-1999-01-5578, Ocotober 1999.

[25] M. A. Hale and J. I. Craig. Techniques for Integrating Computer Programs into Design Architectures. In *6th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, number AIAA-96-4166, September 1996.

[26] M. A. Hale and D. N. Mavris. A Lean-Server Foundation for Collaborative Design. In *5th NASA National Symposium on Large-Scale Analysis, Design and Intelligent Synthesis Environments*, Ocotber 1999.

[27] JMP - Statistical Discovery - Data Analysis Software - Six Sigma - Design of Experiments - Statistics Software Homepage. `http://www.jmpdiscovery.com/`, August 2002.

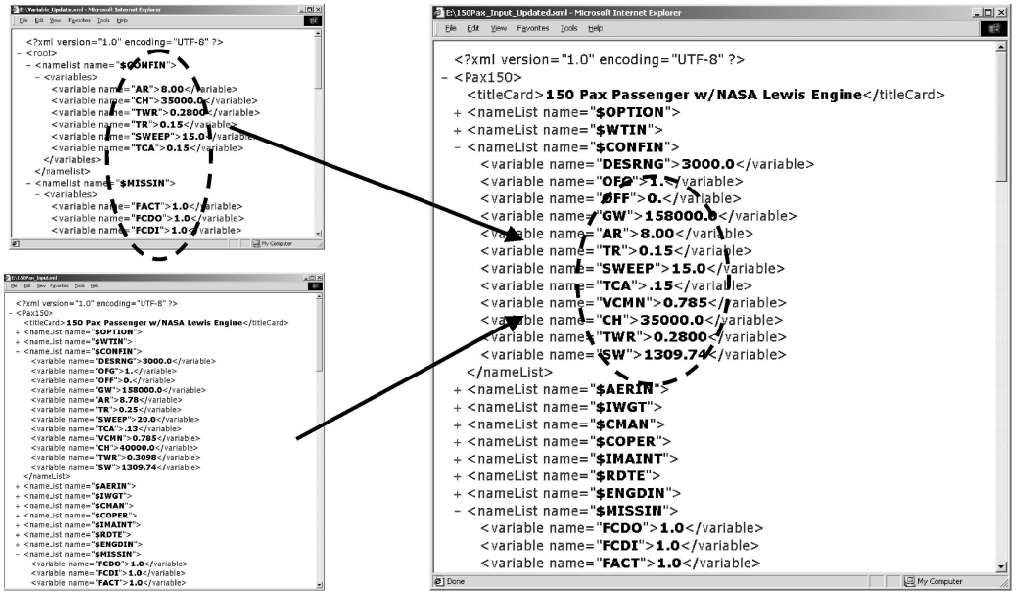

Figure 8: FLOPS Variables Updated with Specific DOE Run Settings

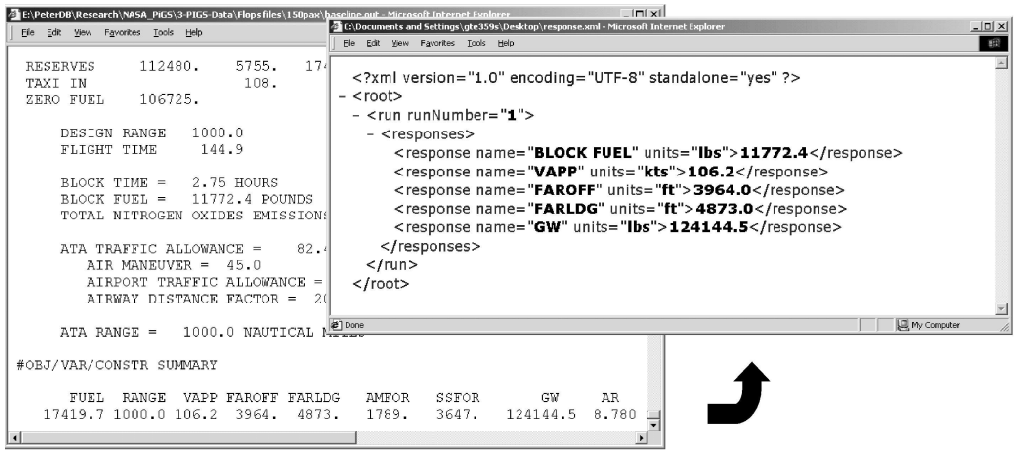

Figure 9: FLOPS Responses Converted from text to XML Format

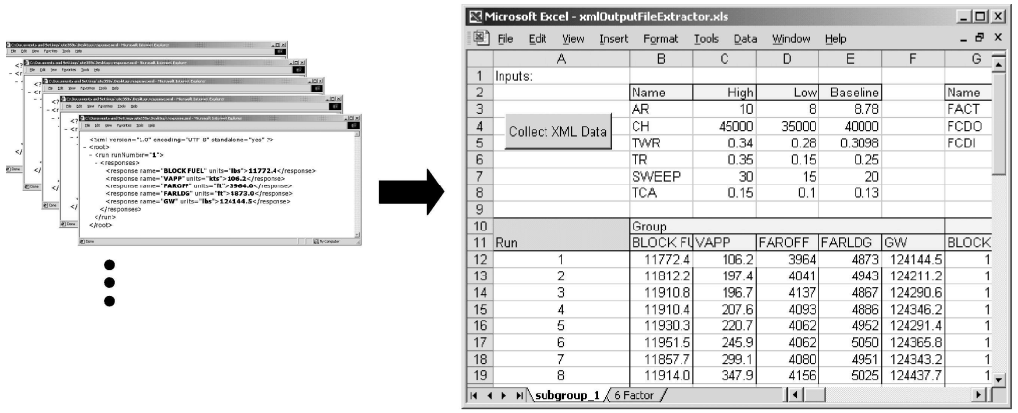

Figure 10: FLOPS Responses Converted from XML Format to Excel